Universitat de Girona
Escola Politècnica Superior

Pràctiques ESII

Curs 2023/24

GEINF- GDDV

Pràctica 3

Grup T
Jordi Badia, Aniol Juanola, Guillem Vidal

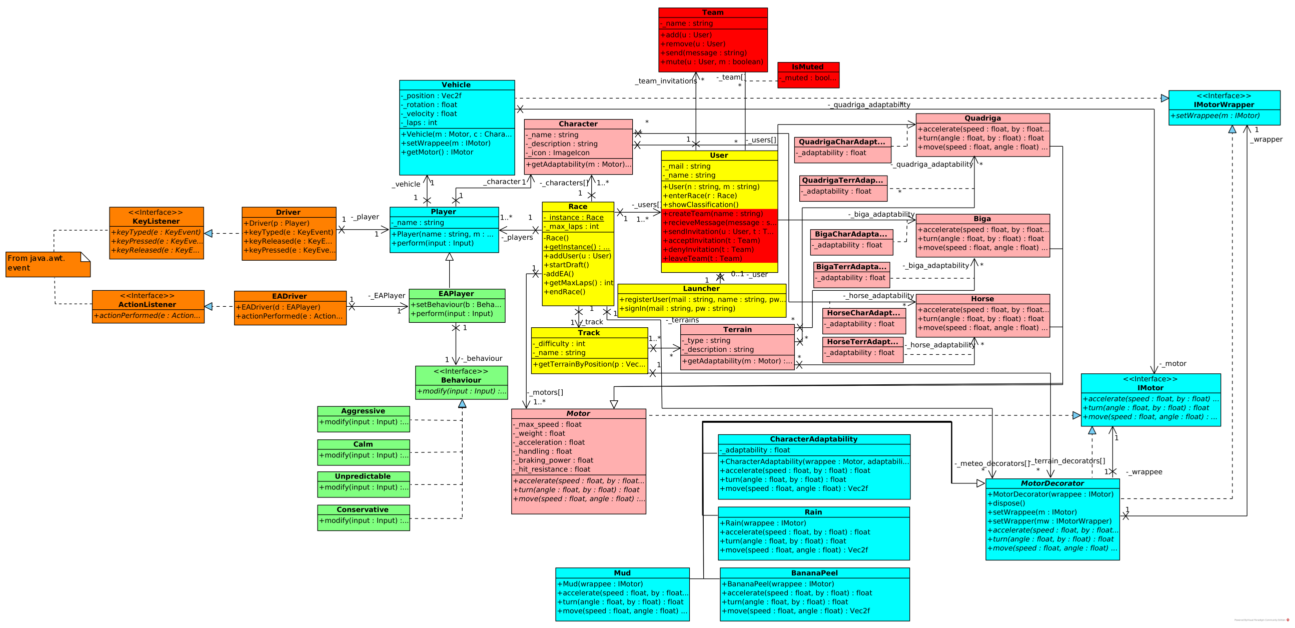
(November 25, 2023)

Contents

1 *Observer* analysis

When implementing the new requirement of sending messages between teams, the direct approach is to go for an observer. Users subscribe to teams and are able to send messages to the group chat, then the team notifies all the subscribed users of this message. See Listing 1.

2 Class diagram



3 *Observer* example

3.1 Problem description

The following example resembles how an online blog would operate. There is one blog which contains an array of posts. These posts simply consist of a `String` (which is the message) and an association with the user who created them. The users can subscribe to up to three observers of blog posts:

NewsBoxNotifier This would represent the typical notification inbox that one would have on a website. It could be interpreted as a PUSH notification on a regular phone.

EmailNotifier This would be an email notification sent to every user's email address.

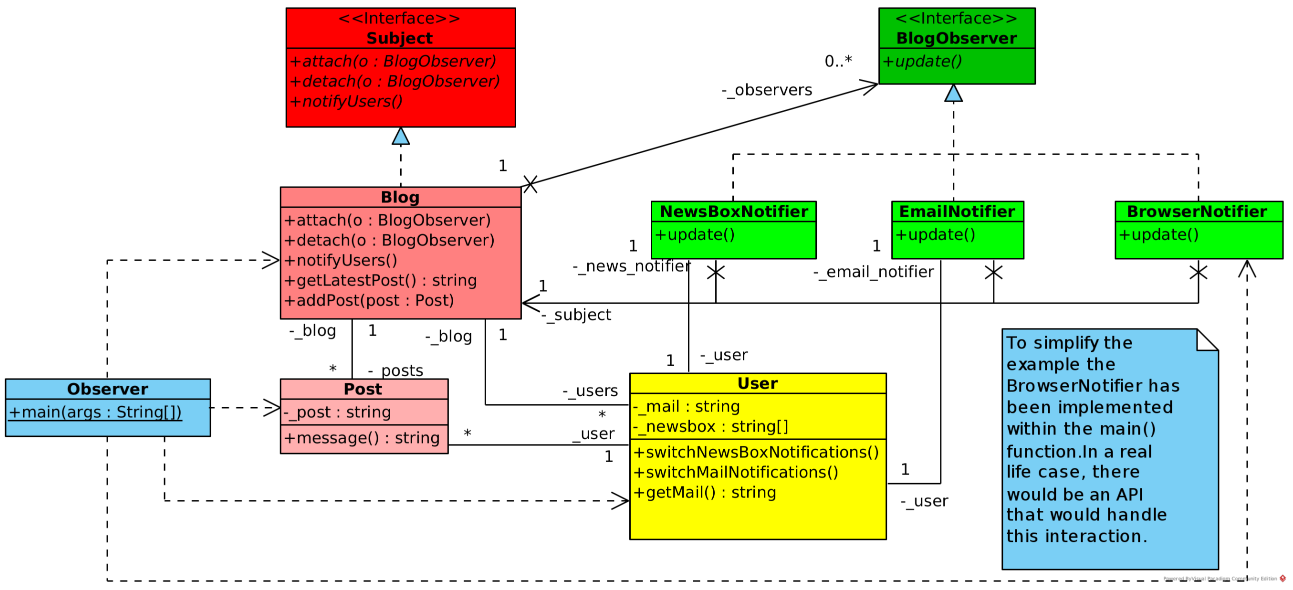
BrowserNotifier This would be the notification that your internet browser pushes. Since this would be handled by an API, we have simply implemented the Observer on the `main()` part of the code. The purpose of the observer can be seen more clearly this way.

The users have the ability to turn on and off the NewsBox and Email notifications (the browser ones would be handled by the browser itself) whenever they want. The notifications show the text of the post up to 10 characters, to emulate the fact that the users would have to click the notification to read the whole message. The example consists of three users activating and deactivating their notifications and posts being posted by them in between these actions. The outputs of the `notify()` function are simply printed on screen.

```java
public class Team {

    private ArrayList<User> users;

    public void add(User user) {

        users.add(user);

    }

    public void send(String message) {

        for (User user : users) {

            user.receiveMessage(message);

        }

    }

}

public class User {

    public void receiveMessage(String message) {

        // Show somewhere...

    }

}

Team team = User.createTeam("Team");

team.add(me);
User.sendInvitation(friend, team);

team.send("Hello, friends!");

/*
 * If the user 'friend' accepted the invitation before sending the message,
 * he will receive the "Hello, friends!" message
 */

team.mute(me, true);

/* User 'me' will no longer receive messages from team "Team" */
```

Listing 1: Usage and general description of the Observer added to implement Team messaging.

3.2 Class diagram


<<Interface>>
Subject
+attach(o : BlogObserver)
+detach(o : BlogObserver)
+notifyUsers()
<<Interface>>
BlogObserver
+update()
0..*
-_observers
1
Blog
+attach(o : BlogObserver)
+detach(o : BlogObserver)
+notifyUsers()
+getLatestPost() : string
+addPost(post : Post)
NewsBoxNotifier
+update()
EmailNotifier
+update()
BrowserNotifier
+update()
1
-_news_notifier
1
-_email_notifier
1
-_subject
-_blog
1
-_blog
1
1
-_user
*
-_posts
Observer
+main(args : String[])
Post
-_post : string
+message() : string
-_users
*
*
_user
1
User
-_mail : string
-_newsbox : string[]
+switchNewsBoxNotifications()
+switchMailNotifications()
+getMail() : string
1
-_user
To simplify the example the BrowserNotifier has been implemented within the main() function.In a real life case, there would be an API that would handle this interaction.